НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: «Моделювання автоматизованої системи логістики транспортних засобів»

Виконав: студент IV курсу, групи КВ-51

Сонін Олег Валентинович _____
(підпис)

Керівник асистент каф. СПіСКС Радченко К.О. _____
(підпис)

Консультант кандидат технічних наук, доцент, Клятченко Я.М. _____
(підпис)

Рецензент кандидат технічних наук, доцент, Марковський О.П. _____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Пояснювальна записка
до дипломного проекту

на тему: «Моделювання автоматизованої системи логістики транспортних засобів»

Київ – 2019 року

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045440.002 ТЗ	Моделювання	4		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Технічне завдання.			
	А4	ІАЛЦ.045440.003 ТП	Моделювання	2		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Відомість технічного			
			проекту.			
	А4	ІАЛЦ.045440.004 ПЗ	Моделювання	51		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Пояснювальна записка.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.001 ОА		
Розробив		Сонін О.В.			Моделювання автоматизованої системи керування транспортними засобами Опис альбому	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Радченко К.О.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-51	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.005 Д1	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема загальної архітектури			
			програми.			
	А4	ІАЛЦ.045440.006 Д2	Моделювання			
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема бази даних.			
	А4	ІАЛЦ.045440.007 Д3	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Блок-схема алгоритму			
			Дейкстри			
	А4	ІАЛЦ.045440.008 Д4	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема роботи GPS у системі			
			логістики			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			
			Презентація.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.001 ОА	Арк. 3

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Моделювання автоматизованої системи логістики транспортних засобів».

Галузь застосування: проектування автоматизованої системи пошуку оптимального маршруту транспортних засобів.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення програми, яка б забезпечила візуальне моделювання роботи автоматизованої системи логістики транспортних засобів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з будь якою операційною системою (Windows, Ubuntu, Mac OS та інші);
- створювати різні мапи, якими будуть пересуватися автомобілі

- моделювати випадковий набір початкових та кінцевих координат, випадкову кількість елементів моделі
- власноруч задавати ці параметри для більш детального аналізу поведінки системи логістики.

5.2. Вимоги до апаратного забезпечення

- Процесор: 2,4-ядерний, Intel Core i7, i5, i3;
- Оперативна пам’ять: 8 Гб;
- Наявність доступу до мережі Internet;

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows 8+, Ubuntu 16.04+, Mac OS та інші;
- Браузер Google Chrome, Mozilla Firefox, Opera та інші;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.04.2019
2.	Розроблення та узгодження технічного завдання	30.04.2019
3.	Аналіз існуючих рішень	05.05.2019
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2019
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2019
6.	Підготовка графічної частини дипломного проекту	20.05.2019
7.	Оформлення документації дипломного проекту	25.05.2019
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045440.004 ПЗ	Моделювання	51		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Пояснювальна записка.			
	А4	ІАЛЦ.045440.005 Д1	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема загальної			
			програми.			
	А4	ІАЛЦ.045440.006 Д2	Моделювання			
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема бази даних.			

					ІАЛЦ.045440.003 ТП
Змін.	Арк.	№ докум.	Підпис	Дата	
Розробив		Сонін О.В.			Моделювання автоматизованої системи керування транспортними засобами Відомість технічного проекту
Перевірив		Радченко К.О.			
Н. контроль		Клятченко Я.М.			
Затвердив		Тарасенко В.П.			

Літ.	Аркуш	Аркушів
	1	2

КПІ ім. Ігоря Сікорського, ФПМ, КВ-51

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.007 Д3	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Блок-схема алгоритму			
			Дейкстри			
	А4	ІАЛЦ.045440.008 Д4	Моделювання	1		
			автоматизованої системи			
			керування транспортними			
			засобами.			
			Схема роботи GPS у			
			логістики			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			
			Презентація.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.003 ТП	Арк. 2

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних.

ПЗ – програмне забезпечення.

MVC – шаблон архітектури програми у якому робота з базою даних, представлення користувацького інтерфейсу, та обробка запитів користувача та формування представлення розділені на різні модулі.

PHP – скриптова мова програмування веб-додатків.

ВСТУП

За останні десятки років людство зробило величезний стрибок у своєму розвитку, бажаючи зменшити свою залученість у рутинної ручної праці. Зокрема, у транспортній сфері вже безліч винаходів та патентів, що дозволяють використовувати комп'ютерні технології для полегшення керування автомобілем, такі як: пристрої для контролю за дорожньою розміткою, автопілот, програмне керування максимальною швидкістю, що дозволена в тій, чи іншій країні, автоматичне гальмування у разі виявлення попереду перешкоджаючих предметів чи істот. Також для зручності водіїв на заміну паперовим мапам місцевості прийшли навігаційні пристрої.

Навігація – це галузь знань про спрямування об'єкту до пункту призначення за оптимальним маршрутом та з мінімальною тривалістю необхідної подорожі.

GPS-навігація (англійською Global Positioning System – система глобального позиціонування) – це спрямування об'єкту до пункту призначення з використанням супутникової системи формулювання координат. Вона функціонує двадцять чотири години на добу, за будь-яких погодних умов, та дозволяє сформувати розуміння поточного місця розташування, а також швидкість та напрямок руху об'єкта. Ця система утворена з двадцяти чотирьох супутників, що обертаються навколо Землі по шести різним орбітам. Визначення географічних координат, а також висоти над рівнем моря приймача відбувається за допомогою вимірювання часу, який витрачається на відправку та прийом сигналу від супутників.

Маючи координати GPS-приймача навігаційне ПЗ здатне прокласти найкоротший маршрут до точки призначення, слідкувати і, якщо з'являється необхідність, повідомляти водієві про відхилення від запланованого маршруту.

Сучасне навігаційне ПЗ оброблює приблизну інформацію про стан завантаженості деяких ділянок автомобільних доріг, ремонтні роботи, аварії, та

надає можливість водіям спілкуватися один з одним про перешкоди, які є на ділянці дороги, на якій вони зараз перебувають, або планують проїхати.

Якщо ж об’єднати усі транспортні засоби в одну систему і кожному транспортному засобу визначати маршрут, яким він має рухатись, то можна аналізувати набагато більше постійно змінюваних параметрів для забезпечення максимальної оптимізації маршруту кожного транспортного засобу.

1. АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ТА СИСТЕМ ОБРОБКИ ДАНИХ

1.1. Задача про найкоротший шлях

В теорії графів найбільш ґрунтовною є задача про визначення найкоротшого шляху між декількома вузлами графу. Застосовується ця задача у багатьох сферах людської діяльності, зокрема, у пересуванні на місцевості. В такому випадку вузлами графу стають перехрестя, вигини дороги, а ось ребрами стають проміжки дороги між перехрестями та вигинами. Вагою кожного такого ребра є кількість часу, необхідна для його повного проходження, або довжина цього проміжку дороги.

Існує декілька типів згадок про цю задачу, щоб відокремити від таких узагальнень:

- Задача про найкоротші маршрути з однією стартовою вершиною. У цій задачі потребується відшукати найкоротші маршрути від цієї вершини до кожної іншої вершини необхідного.
- Задача про найкоротші маршрути з однією кінцевою вершиною. Ця задача полягає у пошуку найкоротшого маршруту від кожної вершини графа окрім заданої до кінцевої, якою завершується прямування. Ця задача ідентична попередній, та можна її представити у вигляді попередньої інвертувавши ребра графа.
- Задача про найкоротші маршрути для всіх вузлів графа. Ця задача потребує пошуку найкоротших маршрутів для усіх можливих комбінацій вузлів один з одним.

Такі узагальнення є більш легкими для створення порядку дій на відміну від циклічного проходження між кожною комбінацією з двох вузлів графа для пошуку маршрутів з заданими параметрами.

1.2. Алгоритми знаходження найкоротшого шляху

Ключовими алгоритмами вирішення задачі є:

- Алгоритм Дейкстри, що працює з ситуаціями, де визначені початкові та кінцеві вузли.
- Алгоритм Беллмана-Форда, що працює з ситуаціями як в алгоритмі Дейкстри, але ребра графа можуть приймати значення ваги менше 0.
- Алгоритм пошуку A*, що працює з ситуаціями, де визначені стартові та кінцеві вузли, та застосовує поступове спрощення для зменшення часу роботи алгоритму.
- Алгоритм Флойда-Уоршелла, що працює з ситуаціями де треба відшукати усі можливі найкоротші маршрути.
- Алгоритм Джонсона, що працює з ситуаціями де треба відшукати усі можливі найкоротші маршрути. Цей алгоритм має сенс використовувати у ситуаціях з більш розрідженими графами.
- Теорія збурень, що працює з ситуаціями у яких потребується відшукати найкоротший маршрут на невеликих розмірах місцевості.

1.2.1. Алгоритм Дейкстри

Алгоритм Дейкстри — алгоритм, винайдений Едсгером Вібе Дейкстрою, який здійснює пошук найкоротшого маршруту від однієї вершини графа до тих, які залишилися. Алгоритм працює лише для графів без циклів, а також тільки з довжинами ребер не менше нуля.

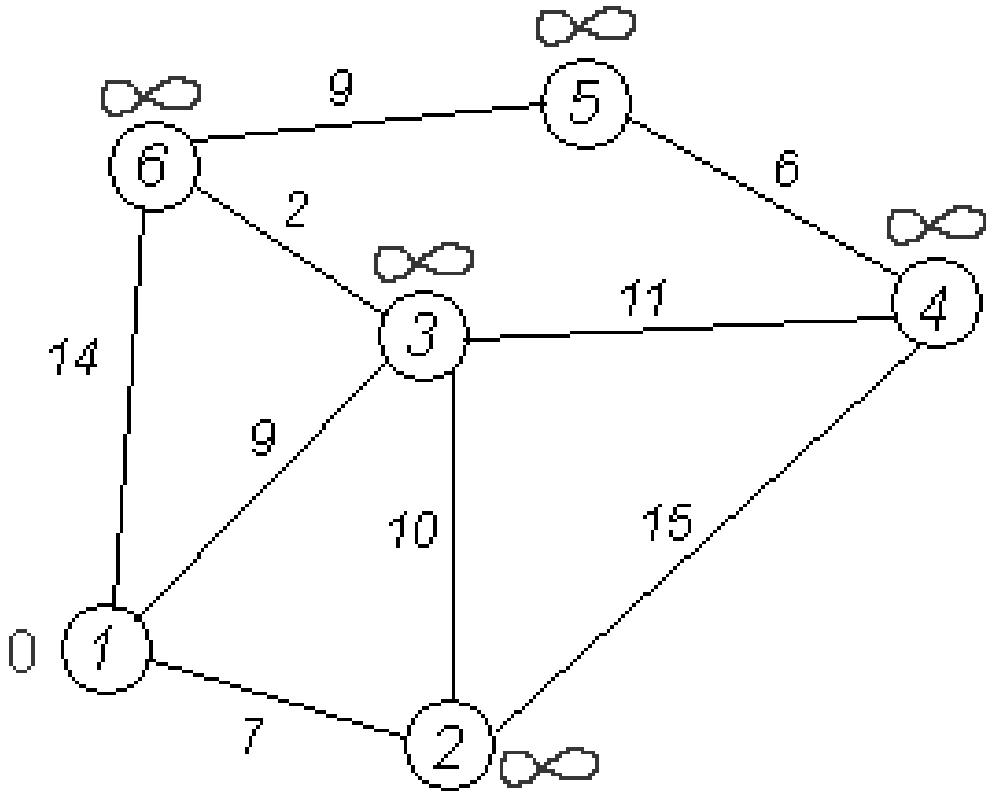


Рисунок 1.1 – Граф для пояснення алгоритму Дейкстри

Принцип функціонування цього алгоритму полягає у наступному. Визначається поточну найменша відстань до всіх вершин графа від вхідного вузла a, та в процесі роботи алгоритму потребується спробувати зробити це значення менше. На етапі ініціалізації усі значення ребер розраховуються як нескінченність, окрім відстані до вхідної точки, яка позначається нулем.

На рисунку 1.1 Над ребрами написана відстань між його кінцевими вузлами. Біля вузлів написана поточний найкоротший маршрут до цього вузла.

- Крок 1. Ініціалізація. Довжина усіх ребер – нескінченність. Довжина до вхідного вузла дорівнює нулю.
- Крок 2. Обираємо вузел, де поточний найкоротший маршрут з якого найменший. В прикладі вище це вузел під номером 1. Циклічно перебираємо кожен вузел, що має спільне ребро з обраним, та у випадку коли значення відстані не більше та не дорівнює

поточному найменшому значенню маршруту до цього вузла, то зберігаємо саме цей.

- Крок 3. Перший суміжний вузел до 1-го це вузел під номером 2. Маршрут до 2 вузла складається з відстані до 1 вузла, ребра між 1 та 2 вузлом. Отже маємо маршрут з довжиною 7, який, очевидно є коротшим за встановлений зараз на етапі ініціалізації, тому відмічаємо цей маршрут як мінімальний.
- Крок 4. Попередній крок повторюється аналогічно для усіх вузлів, суміжних з 1 вузлом, що залишилися.
- Крок 5. Таким чином маємо ситуацію, коли всі суміжні вузли до 1 вузла пройдені. Визначена вхідна вершина має найменшу відстань сама до себе, тому можна її викреслити з подальшого її аналізування.
- Крок 6. Далі треба знайти вузел з найменшим значенням маршруту до нього для подальшої роботи алгоритму, яка ще не викреслена для обробки (не пройдена). У даному випадку це вузел під номером 2, значення маршруту до якого дорівнює 7.
- Крок 7. Тепер треба спробувати знайти маршрут до обраного вузла, значення якого буде меншим за 7. Для цього аналізуємо вузли, що є суміжними з цим вузлом. Такими вузлами є вузли під номерами 1, 3, 4.
- Крок 8. Оскільки на шостому кроці перша вершина була викреслена (пройдена), та маршрут з неї до поточного вузла вже відомий, то з нею нічого не слід робити.
- Крок 9. Наступний за порядком вузел, суміжний з 2 – це вузел 3. Маршрут до нього через вузел 2 складається з вузла 1, ребра між 1 і 2 вузлом довжиною 7, та ребра між 2 та 3 вузлом довжиною 10. Отже цей маршрут має сумарну довжину 17, що значно більше ніж

поточно визначена довжина маршруту до 3 вузла через вузел 1, тому цей маршрут не є оптимальним і довжина 10 не змінюється на 17.

- Крок 10. Усі суміжні вузли для 2 вузла перевірені, тому вважаємо цю вершину пройденою (проаналізованою) і викреслюємо її, та поки що зберігаємо значення довжини маршруту до неї.
- Крок 11. Проводимо аналогічні операції з кроку 2 до 6 і визначаємо, що вузел 3 є найближчим.
- Наступні кроки. Проводимо ті ж самі операції для вузлів, що залишилися, а саме 4, 5, та 6.
- Завершення роботи алгоритму. Алгоритм припиняє працювати, коли викреслений кожний вузол. Остаточний результат з довжинами маршрутів до кожного вузла на рисунку 1.2.

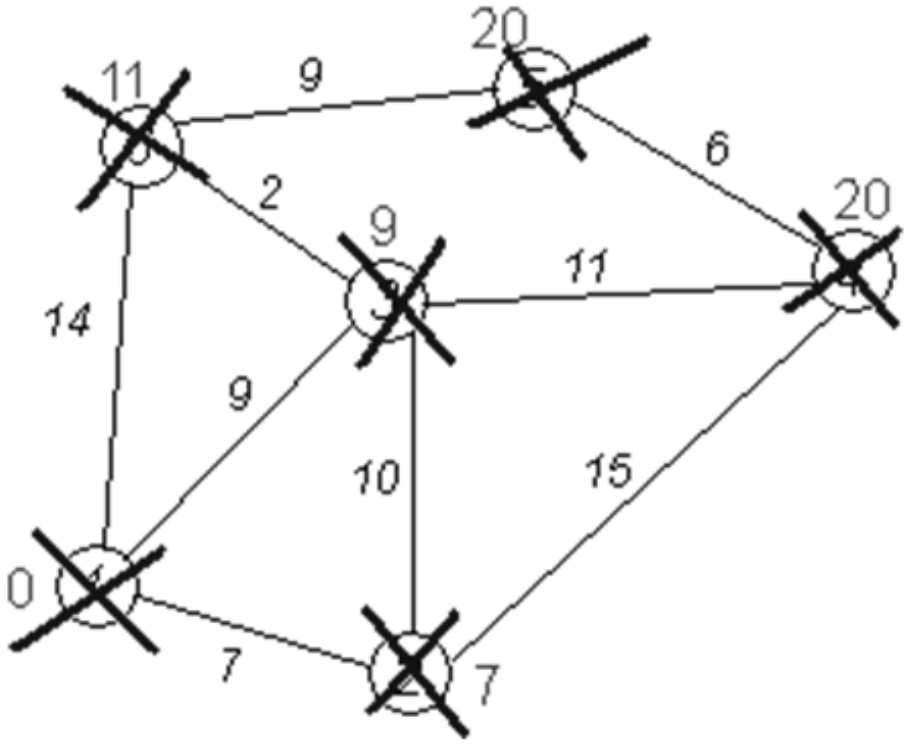


Рисунок 1.2 – Граф після закінчення роботи алгоритму Дейкстри

1.2.2. Алгоритм Беллмана—Форда

Алгоритм Беллмана—Форда — алгоритм пошуку найкоротшого шляху в зваженому графі. Знаходить найкоротші шляхи від однієї вершини графа до всіх інших. На відміну від алгоритму Дейкстри, алгоритм Беллмана—Форда допускає ребра з негативною вагою. Запропоновано незалежно Річардом Беллманом і Лестером Фордом [3].

1.2.3. Алгоритм пошуку A*

Алгоритм пошуку A* — належить до евристичних алгоритмів пошуку. Використовується для пошуку найкоротшого маршруту між двома вершинами графу з додатніми вагами ребер. Описаний 1968 р. Пітером Хартом, Нільсом Нільсоном та Бертрамом Рафаелєм [4].

Алгоритм розрізняє такі групи вузлів:

- незнайомі: ці вузли ще не виявлені. Ще не розрахований маршрут до них. На старті роботи алгоритму всі вузли, без початкової, припадають до групи незнайомих.
- Знайомі вузли: вже знайомий (але ймовірно не найменший) маршрут до цих вузлів. Всі відомі вузли зберігаються в перелік. З переліку відбираються, перш за все, багатообіцяючі вузли. Виконування цього переліку суттєво позначається на темп алгоритму, і розуміється, має подобу черги з лідерством.
- остаточно опрацьовані вузли: ці вузли мають прорахований мінімальний маршрут. Остаточно опрацьовані вузли приєднуються остаточного переліку, щоб попередити багатократному цих вузлів. Остаточно опрацьовані вузли приєднуються остаточного переліку, щоб попередити багатократному цих вузлів. Перелік остаточно опрацьованих вузлів формується в процесі, та на старті у ньому немає записів.

Кожний остаточно опрацьований вузел вбачає певний ідентифікатор минулого вузла. Внаслідок чого з’являється можливість відтворити повний маршрут.

Коли поточний вузел стає остаточно опрацьованим, сусідні вузли вносяться до переліку знайомих вузлів, а поточний вузел вноситься до переліку остаточно опрацьованих вузлів. Ідентифікатор минулого вузла налаштовується на поточний вузел.

Сусідні вузли, які перебувають в переліку остаточно опрацьованих вузлів, до переліку знайомих не вносяться, а ідентифікатори залишаються незмінними. Сусідні вузли, що перебувають в переліку знайомих вузлів, тільки актуалізуються (значення та ідентифікатор минулого вузла), коли прорахований до них маршрут менший за поточний маршрут. Алгоритм завершує роботу якщо фінальний вузел вноситься до переліку остаточно опрацьованих вузлів. Прорахований маршрут представляється з використанням ідентифікаторів минулих вузлів. У разі, коли перелік знайомих вузлів стає пустим, можливості розрахувати маршрут немає.

Представлений прорахований за ідентифікаторами минулих вузлів маршрут відображається з хвоста маршруту, тобто спочатку йдуть фінальні вузли. Задля того, щоб миттєво відобразити маршрут в істинному спрямуванні, зі стартового вузла до фінального, цей маршрут проходить інверсію.

1.2.4. Алгоритм Джонсона

Алгоритм Джонсона дозволяє знайти найкоротші шляхи між усіма парами вершин зваженого орієнтованого графа. Цей алгоритм працює, якщо у графі містяться ребра з додатною чи від'ємною вагою, але відсутні цикли з від'ємною вагою [6].

В алгоритмі Джонсона використовується алгоритм Беллмана-Форда і алгоритм Дейкстри, реалізовані у вигляді підпрограм. Ребра зберігаються у вигляді списків суміжних вершин. Алгоритм повертає звичайну квадратну

матрицю з заданим розміром, або видає повідомлення про те, що вхідний граф містить цикл з негативною вагою [6].

Псевдокод алгоритму Джонсона представлений на рис. 1.3

```
Будується граф

 if Bellman_Ford      = FALSE
    then print «Вхідний граф містить цикл з негативною вагою»

    else'for' для кожної

         do присвоїти величині      значення     ,
            обчислене алгоритмом Беллмана - Форда

         for для кожного ребра

             do

         for для кожної вершини
             do обчислення за допомогою алгоритму Дейкстри

                  величин

             для всіх вершин

             for для кожної вершини

                 do
    return D
```

Рисунок 1.3 – Псевдокод алгоритму Джонсона

1.3. Існуючі аналоги

До існуючих аналогів можна віднести такі сервіси, як EasyWay, Google Maps, OpenStreetMap, та ін. Такі системи дозволяють розрахувати маршрут руху від точки А до Б, але не враховують зміни різних параметрів у реальному часі, тому можливі ситуації, коли ці сервіси направляють автомобіль на ділянку дороги, де кілька хвилин тому відбулася велика аварія і перекриті усі полоси.

EasyWay — програма, що представляє можливість наочно побачити інформацію про рейси та зупинки громадського транспорту в Україні, Молдові, Болгарії, Росії, Польщі та інших країнах.

Сервіс EasyWay надає можливість розрахувати потрібний маршрут прислухаючись до параметрів платежеспроможності та необхідної швидкості, до того ж може прорахувати найкоротший автомобільний маршрут.

Сервісом EasyWay розроблене ПЗ для мобільних операційних систем Android та iOS, та надається API доступ.

Це ПЗ розрахунку шляхів громадського транспорту являє собою соціальний проект, що не потребує купівлі для використання, замість того використовується інтеграція рекламних блоків, завдяки чому відбувається монетизація та подальший розвиток ПЗ.

OpenStreetMap — це проект з відкритим програмним кодом (Open Source) що збирає, запам'ятовує та поширює загальнодоступну картографічну інформацію, виготовляє знаряддя для обробки цієї інформації використовуючи наробки Open Source спільноти.

OpenStreetMap проект створений у Англії в липні 2004 року.

Автор проекту – Стів Кост.

Карти Google — картографічний сервіс від компанії Google що не потребує оплати за користування, як і OpenStreetMap та EasyWay, а також набір розробленого ПЗ.

Основна розробка – це веб-додаток, що містить карту та світлини з усіх куточків планети Земля, а також Марсу, та супутника – Місяця, які надають людству можливість продивлятися вулиці, прокладати маршрут (автомобілем, пішки, велосипедом або громадським транспортом). З сервісом взаємодіє карта автомобільних доріг, з пошуком маршрутів.

Перегляд світлин з планет може бути здійснений в декількох режимах: «зверху-вниз», «польоту». Більшість світлин високої роздільної здатності створені з використанням квадрокоптерів, що рухаються на висоті 240—460 м над поверхнею Землі, інші світлини зроблені камерами супутників. Кожна світлина оновлюється приблизно кожні три роки.

1.4. Обґрунтування теми дипломного проекту

Більшість аналогів створюють оптимальний маршрут руху транспортних засобів аналізуючи лише статичні параметри, такі як довжина маршруту, середня швидкість руху, зовсім не сприймаючи до уваги ситуацію на дорозі в реальному часі. Через це оптимальним маршрутом це назвати неможливо.

Даний проект спрямований на те, щоб створити модель повноцінної автоматизованої системи логістики транспортних засобів, у якій для знаходження оптимального маршруту будуть аналізуватись також і деякі динамічні параметри, такі як: завантаженість ділянки дороги, наявність аварій, середній час очікування увімкнення зеленого світла на перехрестях і т. д.

Для реалізації були обрані мови програмування PHP та Javascript, оскільки вони мають весь необхідний інструментарій для створення моделі автоматизованої системи логістики транспортних засобів і є мовами програмування веб-додатків. Завдяки цьому є можливість переглядати модель на широкому діапазоні пристроїв з різними операційними системами.

2. ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ. РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ЛОГІСТИКИ.

2.1. Опис необхідного для розробки інструментарію

2.1.1. Операційна система

Розробка програмного забезпечення проводилася на комп’ютері з операційною системою компанії Microsoft, що має назву Windows, 10 версії. Оскільки це одна з найпоширеніших ОС у світі, то й різноманітних утиліт та програм для роботи вистачає для комфортної роботи над істотною кількістю задач.

Перші версії Microsoft Windows не були справжніми операційними системами, а лише представляли з себе модулі до ОС MS-DOS.

Мінімальні вимоги до апаратного забезпечення для встановлення Windows 10 такі:

- Тактова частота процесора 1 ГГц;
- Оперативна пам’ять від 1 Гігабайта;
- Відеокарта з підтримкою технології DirectX 9;
- Монітор з роздільною здатністю від 800×600 пікселів;
- Пам’ять на вінчестері від 32 Гб;
- Клавіатура та комп’ютерна миша;

Windows 10 працює на багатьох теперішніх сучасних пристроях (смартфони, планшети, персональні комп'ютери, ноутбуки, навіть телевізори, тощо). Windows Phone не акомпанує з ОС, замість неї розроблено Windows 10 з легким, та менш функціональним стеком технологій для мобільних пристроїв.

Якщо користувач під’єднанав до мобільного пристрою клавіатуру і пристрій графічного виводу, ОС радить мігрувати на повноцінний режим використання ОС.

Якщо пристрій підключений до мережі Інтернет-зв’язку і користувач підключив до ПК пристрій, невідомий для ПЗ чи ОС, Windows створює пошук необхідного драйвера, та інсталює його для подальшої роботи з пристроєм.

2.1.2. Мова програмування

Як вже було зазначено, основною мовою програмування було обрано PHP.

У PHP можлива ситуація, коли посеред коду влаштований html-код веб-сторінок, який опрацовується мовою.

Обробник PHP стартує інтерпретацію команд з коду після відкриваючого ключового слова «<?php» і провадить виконання доки не зустріне закриваюче ключове слово «?>».

Велике розмаїття стандартних функцій дозволяє ухилятись від створення великих неоптимізованих функції, та витрачати на низькорівневу логіку зовсім мало часу, як це відбувається в деяких сучасних мовах програмування.

У PHP налаштовані драйвера для інтеграції з MySQL, PostgreSQL, та багатьох інших сучасних баз даних та технологій.

Мова PHP побудована за популярними принципами, визнаними світовою спільнотою, тому код може виглядати знайомо розробникам, що працюють з різними імперативними мовами програмування. Багато конструкцій мови скопійовані та доповнені з мов C, Perl. Код PHP має спільні риси з типовими програмами, код яких написаний мовами C++ або Javascript. Це відчутно зменшує стартові складнощі під час переходу на PHP. PHP — це мова, яка сполучає сильні сторони мов програмування Perl та C і нарочито розроблена для роботи в мережі Інтернет, має загальновживаний і сприйнятливий синтаксис. І хоча PHP історія знає не довго, вона отримала великий попит web-розробників ПЗ, що зараз вже стала найбільш розповсюдженою у сфері розробки веб-скриптів.

Участь у Open Source, і поширення програмного коду веб-додатків у спільноті веб-розробників, зробили великий ефект на безліч проектів, насамперед — ОС Linux.

Впровадження стратегії Open Source і вільне поширення програмного коду веб-додатків добре послугувало кінцевим користувачам. До того ж, спільнота створює власні середовища для допомоги та підтримки усього різноманіття технологій на базі PHP, та влаштовуються професійні заходи для поширення досвіду розробки веб-скриптів. На багато проблем існує професійна науково обґрунтована відповідь.

Ефективність та швидкодія відносяться до значних факторів у програмуванні для проектів, створених для великої аудиторії кінцевих користувачів, до яких належить і веб. Вагомою плюсом PHP є те, що ця мова є інтерпретованою, що призводить до можливості виконання скриптів з доволі високим темпом.

За деякими оцінками, велика кількість PHP-скриптів, розмір яких не великий, прораховуються набагато швидше на відміну від схожих за складністю програм, що реалізовані на мові Perl.

Продуктивності PHP вистачає для розробки достатньо високонавантажених сервісів, та для комфортного їх використання кінцевими користувачами.

2.1.3. Локальний веб-сервер

Для того щоб запускати PHP-код на комп’ютері, необхідно встановити та налаштувати локальний веб-сервер. Для розробки було використано веб-сервер Open Server.

Open Server – це спеціальний сервер, призначений для веб-розробки, тестування веб-проектів. Він включає в себе комплекс таких програм:

Apache – надійний та гнучкий сервер з відкритим кодом;

Інтерпретатор PHP;

PHPMyAdmin – спеціальний інструмент для роботи з базою даних MySQL, побудований на мові PHP;

Nginx– ще один гнучкий сервер;

MySQL – система, призначення якої – керування базами даних;

та інші.

2.1.4. Середовище розробки та тестування

Для написання програмного коду використовувалося інтегроване середовище розробки PhpStorm від компанії JetBrains. Зразок інтерфейсу представлений на рис. 2.1.

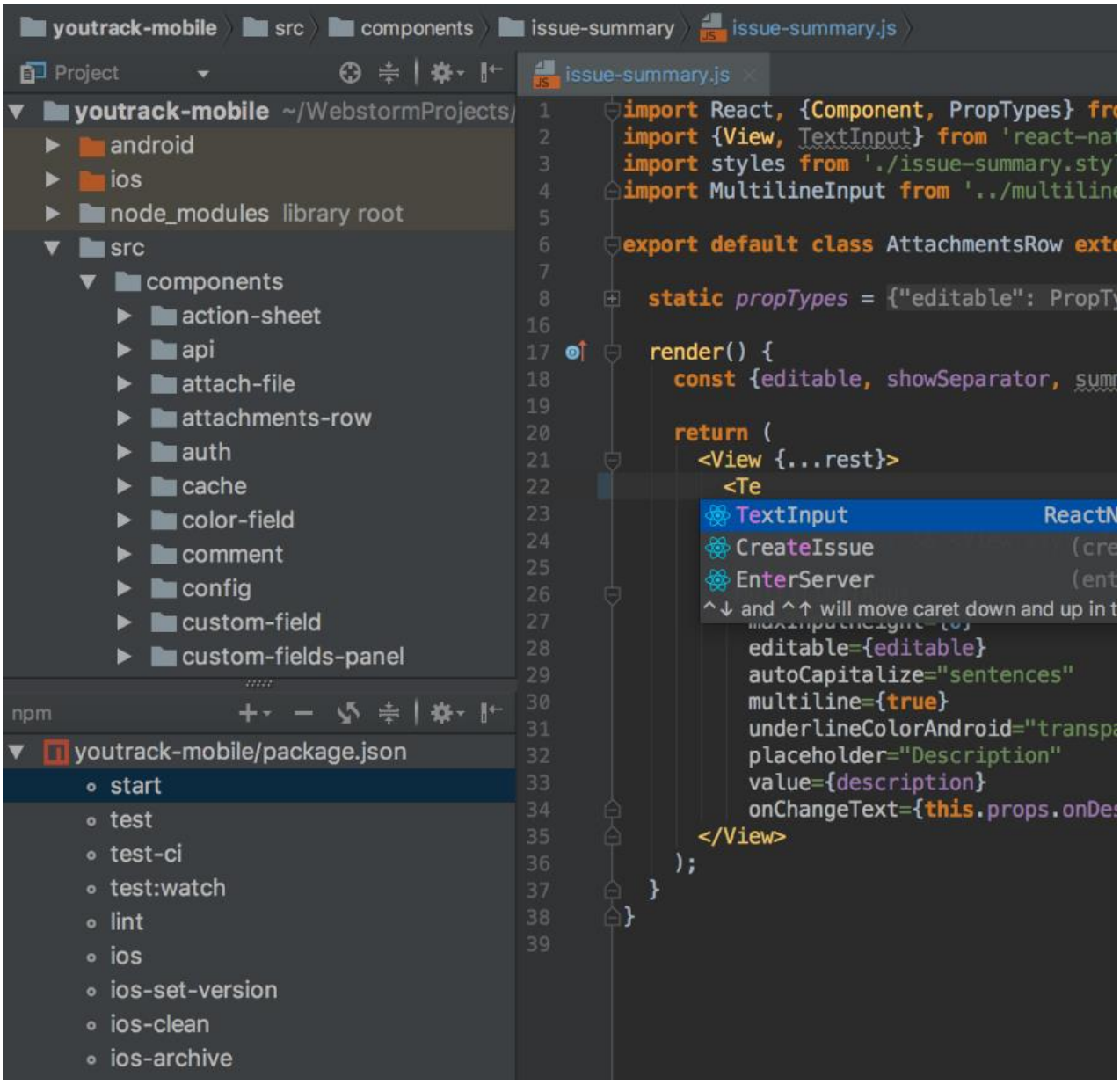


Рисунок 2.1 – Зразок інтерфейсу PhpStorm

PhpStorm представляє собою розумний редагувач для PHP, HTML і JavaScript з використанням алгоритмів аналізу та обробки коду в режимі реального часу, застереження від створення ненавмисних помилок у програмному коді, а також запуск режиму відлагодження програми для PHP і JavaScript.

Автодоповнення коду в PhpStorm засновано на офіційних специфікаціях PHP 5.3/5.4/5.5/5.6/7.0/7.1/7.2/7.3, включаючи назви змінних та функцій, назви класів, модулів, бібліотек, просторів імен, замикать. Наявний також багатофункціональний редактор SQL запитів з можливістю зміни та збереження запитаних результатів запитів, підключені різноманітні модулі для роботи з базами даних, терміналом, та іншими системами.

Оскільки PhpStorm створений на базі IntelliJ IDEA, доступна можливість встановити ще більше плаґінів, які потрібні при розробці дипломного проекту.

2.1.5. Фреймворк

Для полегшення розробки, за рахунок зменшення рутинної роботи використаний фреймворк Laravel.

Laravel — безкоштовний, з відкритим програмним кодом PHP-фреймворк, створений компанією Taylor Otwell та направлений на розробку веб-програм відповідно до шаблону проектування програм model–view–controller (MVC).

Програмний код Laravel зберігається на GitHub та поширюється за умовами ліцензії MIT.

2.2. Загальна архітектура програми

Програма поділяється на три взаємопов'язані частини (рис. 2.2):

- модель даних;
- вигляд (інтерфейс користувача);
- модуль керування.

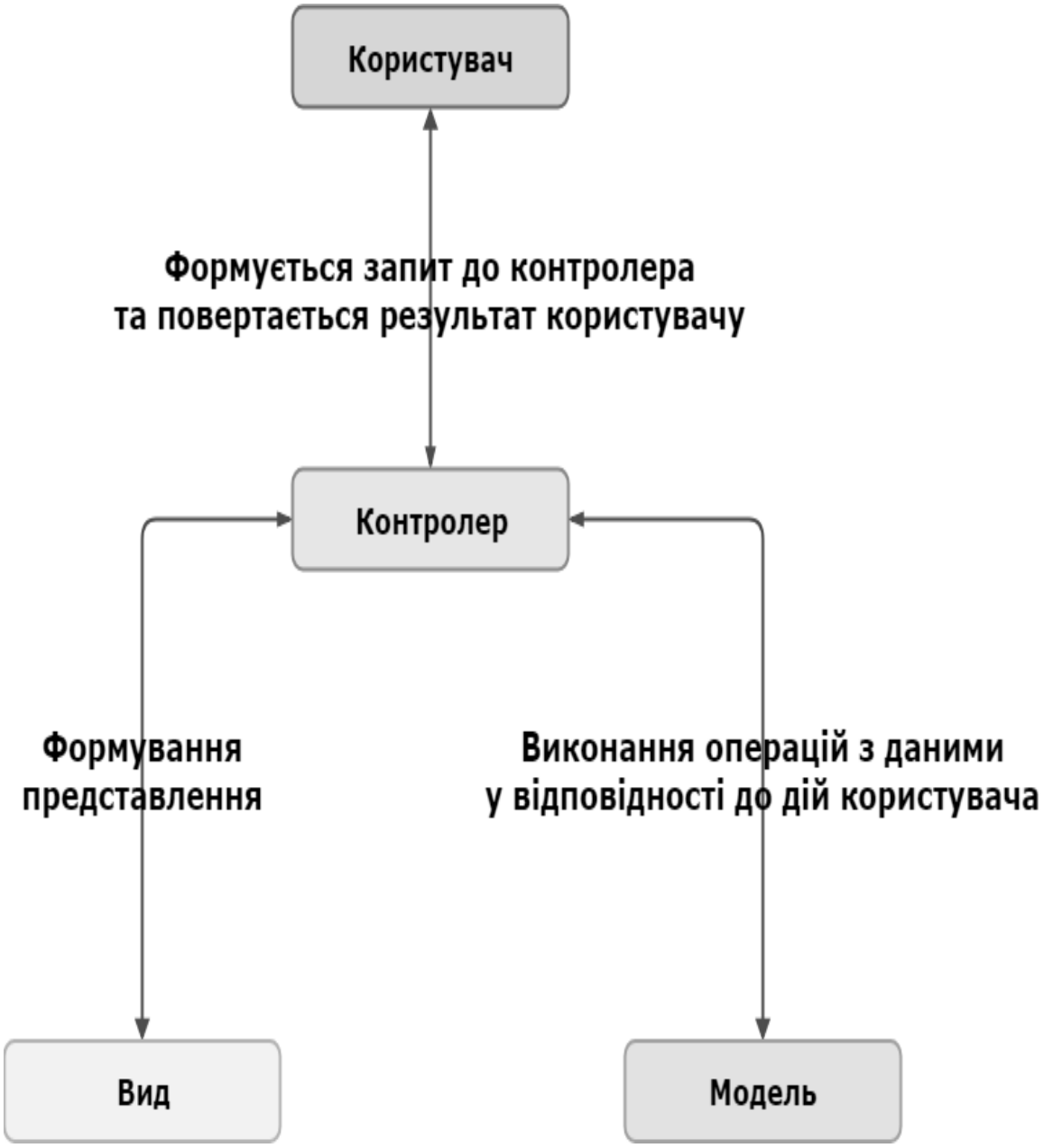


Рисунок 2.2 – Схематичне представлення загальної архітектури програми

Така архітектура забезпечує гнучкий дизайн програмного забезпечення, полегшує подальші зміни чи розширення програм, а також надає можливість повторного використання окремих компонентів програми.

Використання такої архітектури сприяє впорядкованості структури програми і робить її більш зрозумілою за рахунок зменшення складності.

У моделі відбувається обробка команд з контролера для таких задач:

- робота з базою даних
- надсиланню запитів
- створення об’єктів
- редагування об’єктів
- видалення об’єктів
- перевірка вхідних даних
- формування повідомлень про помилкові запити
- формування повідомлень про помилку у заповненні форм, чи неправильному форматі вхідних даних

Представлення вирішує такі задачі:

- формування представлення для користувача
- обробка файлів шаблонів
- компонування шаблонів
- вставка у потрібних місцях блоків з даними з моделі
- вивід на екран повідомлень про помилки
- підключення необхідних файлів стилів та скриптів для анімації об’єктів

Контролер, в свою чергу, виконує наступні дії:

- приймає запити від користувача
- проводить аналіз вхідних даних
- визначає, які необхідно підключити модулі для обробки вхідних даних
- підключає модулі, визначені в попередньому пункті
- відправляє команди на модель
- компонує дані, отримані від моделі
- відправляє представленню команди з формування користувацького інтерфейсу, та передає скомпоновані дані від моделі, для вставки цих даних до веб-сторінки

- отримує сформований користувацький інтерфейс у вигляді html коду
- відправляє відповідь на запит користувача у вигляді html коду, отриманого на попередньому кроці.

2.2.1. Архітектура бази даних

База даних програми складається з шести таблиць, пов'язаних між собою (рис. 2.3):

- Таблиця Maps, яка зберігає інформацію про доступні для моделювання мапи та містить такі поля:
  - MapID – первинний ключ, ідентифікаційний номер мапи;
  - Name – назва мапи;
  - City – назва міста, якому належить мапа;
- Таблиця Nodes, у якій міститься інформація про вузли (перехрестя, світлофори) визначенної мапи.
  - NodeID – первинний ключ, ідентифікаційний номер вузла;
  - MapID – ідентифікаційний номер мапи, якій належить вузел;
  - X – координата X вузла відносно верхнього лівого кута зображення мапи;
  - Y – координата Y вузла;
  - Type – тип вузла (наприклад, паркінг, заправна станція, тощо)
- Таблиця Edges. Ця таблиця призначена для зберігання ребер графу.
  - EdgeID - первинний ключ, ідентифікаційний номер ребра;
  - StartNodeID - ідентифікаційний номер вузла початку ребра;
  - FinishNodeID - ідентифікаційний номер вузла кінця ребра;
  - Length – довжина ребра;
  - maxLoad – максимальна кількість транспортних засобів, які можуть у один проміжок часу пересуватись маршрутом, що проходить через це ребро. Саме це поле впливає на контроль виникнення заторів:

- Status – статус ребра. Якщо ця ділянка дороги повністю перекрита, то записано значення «0», в іншому випадку «1»;
- MapID - ідентифікаційний номер карти;

- Таблиця Cars. У цій таблиці міститься інформація про автомобілі, зареєстровані в системі
  - CarID – первинний ключ, ідентифікатор транспортного засобу;
  - Name – умовна назва;
- Таблиця CarsQueue зберігає початкову та кінцеву точку руху, а також чергу кожного автомобіля, який приймає участь у моделюванні.
  - CarQueueID – первинний ключ, ідентифікатор черги автомобілів;
  - StartNodeID – ідентифікатор початкової точки маршруту поточного транспортного засобу;
  - EndNodeID – ідентифікатор кінцевої точки маршруту поточного транспортного засобу;
  - CarID – ідентифікатор авто;
  - MapID – ідентифікатор мапи;
  - CreatedAt – дата та час додавання автомобіля в чергу
- Таблиця EdgesQueue містить покроковий маршрут для кожного транспортного засобу ж черги CarsQueue.
  - EdgesQueueID – первинний ключ, ідентифікатор кроку маршруту для авто;
  - EdgeID – ідентифікатор ребра, яким має пересуватись авто;
  - CarQueueID – ідентифікатор поїздки для цього авто в поточному моделюванні;
  - MapID – ідентифікатор мапи;
  - CreatedAt – дата та час обрахунку кроку маршруту.

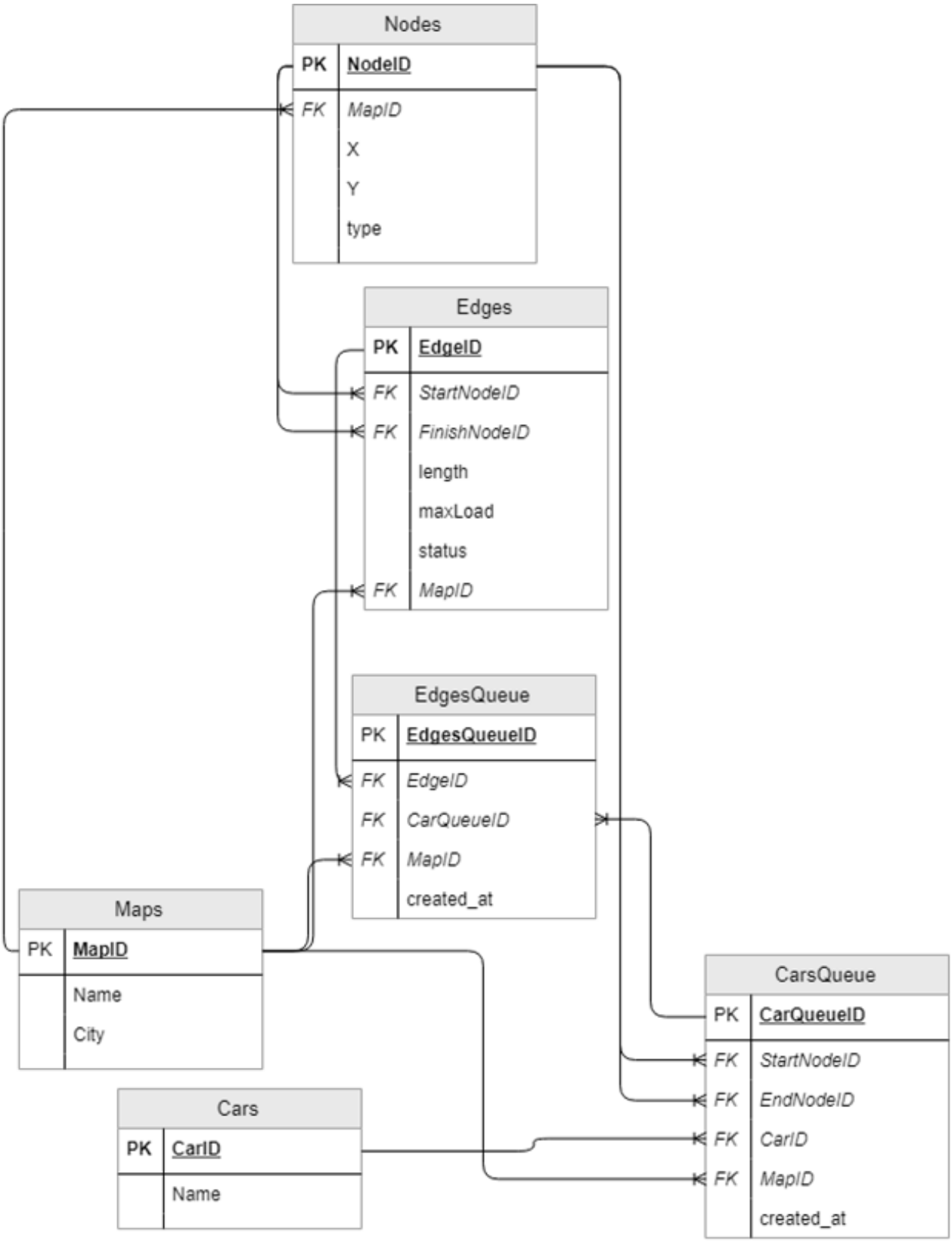


Рисунок 2.3 – Схематичне представлення архітектури бази даних

2.3. Анімація руху

Моделювання руху автомобілів у програмі виконується за допомогою JavaScript, та бібліотеки jQuery Waypoints. Розглянемо детальніше можливості цього функціоналу, та принцип анімації об’єктів.

2.3.1. Принцип роботи jQuery з об’єктами HTML-документу

Для здійснення пошуку по об’єктній моделі документа у jQuery розроблена полегшена можливість створювати пошукові маски – селектори. Ця структура схожа на однойменну в каскадних таблицях стилів (CSS), та мають ширший функціонал. Для прикладу, щоб змінити колір фону в елементі «div», в якого вказаний атрибут «id», треба зробити ініціалізацію jQuery об’єкта з відповідним селектором і запустити функцію зміни фонового кольору (рисунок 2.4):

```
$('div#element').css('background-color', '#aa22tt');
```

Рисунок 2.4 – функція зміни фонового кольору

Що ж до анімації, то для у jQuery є такі типи подій.

Натискання клавіші - подія Click.

Найпростіший спосіб взаємодії з HTML-елементами – це подія Click. Ви натискаєте на щось (наприклад. виділений елемент), і щось відбувається. Click в jQuery працює наступним чином, як на рисунку 2.5

```
$("#target").on( "click", function() {
   alert("Hello");
});
```

Рисунок 2.5 – функція обробки події

Якщо змінна вже оголошена згори сторінки, то можна звернутися до цієї змінної замість того, щоб писати назву селектора (рисунок 2.6).

```
var $clickMeElement = $('#gallery li').eq(0)
$clickMeElement.on( "click", function() {
   alert("Я люблю Codeguida");
});
```

Рисунок 2.6 – приклад звернення до змінної

Це викличе javascript попередження, коли користувач натисне на першу картинку в #gallery.

Переміщення курсору миші - подія Hover.

Зміна поведінки об’єкту в залежності від переміщення курсору миші показана на рисунку 2.7

```
var $hoverMeElement = $('#gallery li').eq(1);
$hoverMeElement.hover(
   function() {
      $(this).css('transform','rotate(90deg)');
   }, function() {
      $(this).css('transform','rotate(0)');
   }
);
```

Рисунок 2.7 – зміна поведінки об’єкту в залежності від курсору миші

Цей фрагмент коду допоможе нам повернути друге зображення в #gallery на 90º при mouseenter і назад (0º) при mouseleave.

Зміна CSS класу при натисканні - ToggleClass подія

Ще одна дуже поширена подія на інтерактивних сайтах, це toggleClass. При виконанні певних дій, jQuery буде додавати або видаляти клас із зазначеного елемента.

`ToggleClass` подія зображена на рисунку 2.8.

```
var $toggleTrigger = $('#gallery li').eq(-1);

$toggleTrigger.click(function() {
  $(this).toggleClass("highlight");
});
```

Рисунок 2.8 – зміна CSS класу

Така дія додасть клас .highlight до елемента #gallery в кінці списку.

2.4. Створення модуля алгоритму Дейкстри

Інтерфейс класу Dijkstra має наступний вигляд (рисунок 2.9)

```php
<?php
 interface Dijkstra {
        var $visited = array();
        var $distance = array();
        var $previousNode = array();
        var $startnode =null;
        var $map = array();
        var $infiniteDistance = 0;
        var $numberOfNodes = 0;
        var $bestPath = 0;
        var $matrixWidth = 0;

        function Dijkstra(&$ourMap, $infiniteDistance) {}

        function findShortestPath($start,$to = null}

        function findBestPath($ourDistance, $ourNodesLeft) {}

        function updateDistanceAndPrevious($obp) {}

        function printMap(&$map) {}

        function getResults($to = null) {}
}
?>
```

Рисунок 2.9 – інтерфейс для класу Dijkstra

Функція Dijkstra виконує ініціалізацію заданої мапи моделі. Реалізація цієї функції має наступний вигляд (рисунок 2.10):

```
function Dijkstra(&$ourMap, $infiniteDistance) {
        $this -> infiniteDistance = $infiniteDistance;
        $this -> map = &$ourMap;
        $this -> numberOfNodes = count($ourMap);
        $this -> bestPath = 0;
}
```

Рисунок 2.10 – ініціалізація об’єкту Dijkstra

Функція ініціалізації задає атрибути класу з даними, що були отримані на вході.

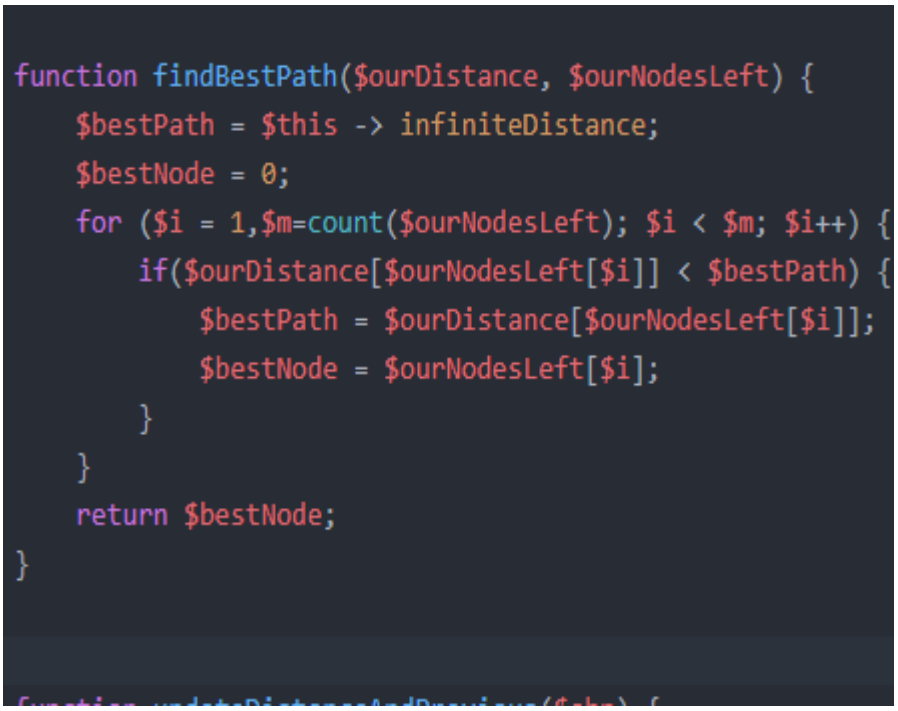
Функція findShortestPath приймає на вході початкову та кінцеву точку маршруту для поточного автомобіля, та циклічно обраховує відстані до кожної сусідньої точки, обираючи з декількох можливих варіантів проїзду найоптимальніший. Реалізація цієї функції подана на рисунку 2.11:

```
function findShortestPath($start,$to = null) {
        $this -> startnode = $start;
        for ($i = 1;$i<$this -> numberOfNodes;$i++) {
                if ($i == $this -> startnode) {
                        $this -> visited[$i] = true;
                        $this -> distance[$i] = 0;
                } else {
                        $this->visited[$i] = false;
                        $this->distance[$i] = isset($this->map[$this->startnode][$i])
                                ? $this -> map[$this -> startnode][$i]
                                : $this -> infiniteDistance;
                }
                $this -> previousNode[$i] = $this -> startnode;
        }

        $maxTries = $this -> numberOfNodes;
        $tries = 0;
        while (in_array(false,$this -> visited,true) && $tries <= $maxTries) {
                $this -> bestPath = $this->findBestPath(
      $this->distance,
      array_keys($this -> visited,false,true)
    );
                if($to !== null && $this -> bestPath == $to) {
                        break;
                }
                $this -> updateDistanceAndPrevious($this -> bestPath);
                $this -> visited[$this -> bestPath] = true;
                $tries++;
        }
}
```

Рисунок 2.11 – реалізація функції findShortestPath

Наступна функція findBestPath (реалізація на рисунку 2.12) – допоміжна до попередньої.



```
function findBestPath($ourDistance, $ourNodesLeft) {
    $bestPath = $this -> infiniteDistance;
    $bestNode = 0;
    for ($i = 1,$m=count($ourNodesLeft); $i < $m; $i++) {
        if($ourDistance[$ourNodesLeft[$i]] < $bestPath) {
            $bestPath = $ourDistance[$ourNodesLeft[$i]];
            $bestNode = $ourNodesLeft[$i];
        }
    }
    return $bestNode;
}
```

Рисунок 2.12 – реалізація функції findBestPath

Ця функція виконує порівняння поточної дистанції з кращою відомою на даний момент і, у разі, якщо поточна дистанція менша за кращу відому, то проходить перезапис і вона відмічається як краща.

Функція updateDistanceAndPrevious (реалізація на рисунку 2.13) переписує значення дистанції поточної та попередньої.

```
function updateDistanceAndPrevious($obp) {
    for ($i = 1;$i<$this -> numberOfNodes;$i++) {
        if(
          isset($this->map[$obp][$i])
          &&  (
            !(
              $this->map[$obp][$i] == $this->infiniteDistance)
              || ($this->map[$obp][$i] == 0 )
            )
          &&  (
                ($this->distance[$obp] + $this->map[$obp][$i])
                < $this -> distance[$i]
              )
        ) {
                $this -> distance[$i] =
                    $this->distance[$obp] + $this->map[$obp][$i];
                $this -> previousNode[$i] = $obp;
        }
    }
}
```

Рисунок 2.13 – реалізація функції updateDistanceAndPrevious

Функція printMap виконує вивід на веб-сторінку мапи у вигляді таблиці відстаней

```php
function printMap(&$map) {
    $placeholder = ' %' . strlen($this->infiniteDistance) . 'd';
    $foo = '';
    for ($i = 1, $im = count($map); $i < $im; $i++) {
        for ($k = 1,$m = $im; $k < $m; $k++) {
            $foo.= sprintf($placeholder,
                isset($map[$i][$k]) ?
                    $map[$i][$k] :
                    $this->infiniteDistance
            );
        }
        $foo .= "\n";
    }
    return $foo;
}
```

Рисунок 2.14 – реалізація функції printMap

Приклад результату роботи попередньої функції представлений на рисунку 2.15

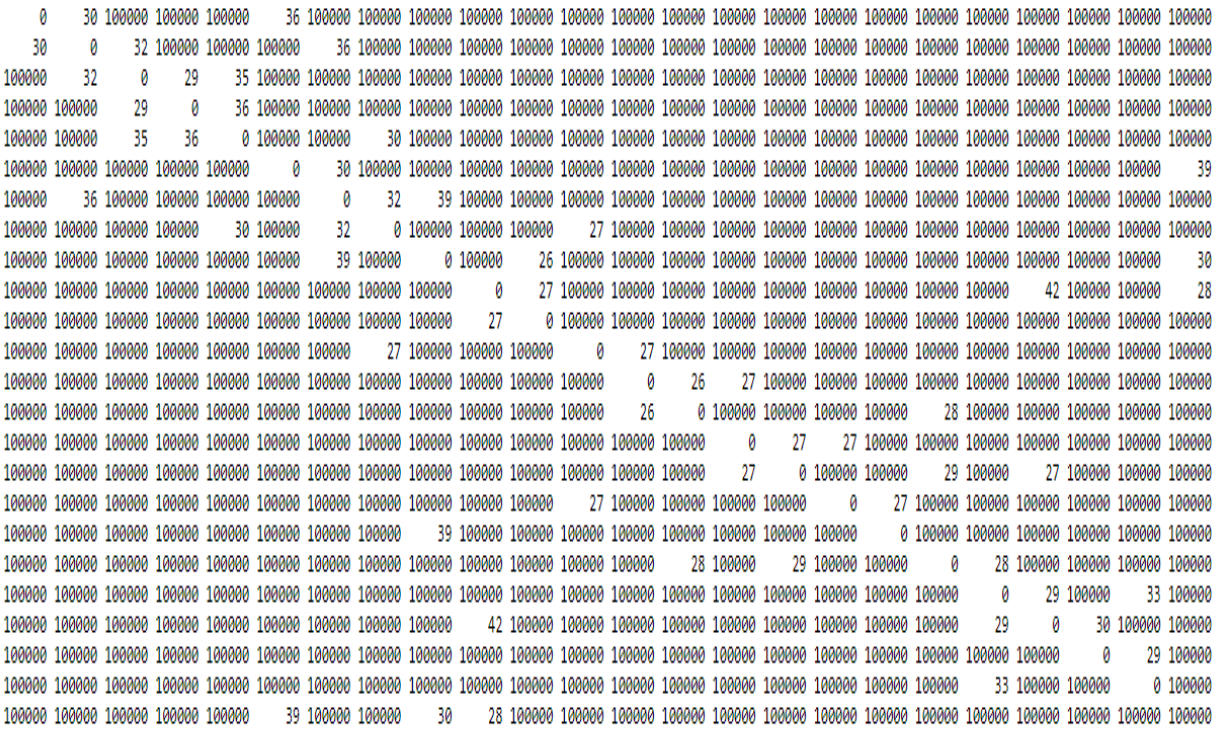


Рисунок 2.15 – Результат роботи функції printMap

На рис. 2.15 часто зустрічається число 100000, яке є емуляцією нескінченної довжини ребра графа, тобто між двома відповідними вузлами немає прямого зв’язку.

Остання в цьому класі функція getResults (реалізація на рисунку 2.16) повертає оптимальний маршрут у вигляді переліку номерів вузлів до одного заданого вузла, або до всіх вузлів поточного графу.

```
function getResults($to = null) {
      $ourShortestPath = array();
      $foo = '';
      for ($i = 1; $i < $this -> numberOfNodes; $i++) {
          if($to !== null && $to !== $i) {
                  continue;
          }

          $ourShortestPath[$i] = array();
          $endNode = null;
          $currNode = $i;
          $ourShortestPath[$i][] = $i;

          while ($endNode === null || $endNode != $this -> startnode) {
                  $ourShortestPath[$i][] = $this->previousNode[$currNode];
                  $endNode = $this->previousNode[$currNode];
                  $currNode = $this->previousNode[$currNode];
          }

          $ourShortestPath[$i] = array_reverse($ourShortestPath[$i]);

          if ($to === null || $to === $i) {
              if($this -> distance[$i] >= $this -> infiniteDistance) {
                  $foo .= sprintf("no route from %d to %d. \n",$this -> startnode,$i);
              } else {
                  $foo .= sprintf('%d => %d = %d [%d]: (%s).'."\n" ,
                      $this -> startnode,$i,($this -> distance[$i]),
                      count($ourShortestPath[$i]),
                      implode('-',$ourShortestPath[$i])
                  );
              }

              $foo .= str_repeat('-',20) . "\n";

              if ($to === $i) {
                  break;
              }
          }
      }
      return $foo;
}
```

Рисунок 2.16 – Результат роботи функції getResults

Зразок вихідних даних цієї функції представлений на рис. 2.17. Спочатку тут друкується початкова та кінцева точка маршруту, потім оцінка оптимальності в даний момент часу, а після цього можна побачити покроковий маршрут між заданими точками.

```
10 => 1 = 167 [6]: (10-24-6-7-2-1).
--------------------
10 => 2 = 136 [5]: (10-24-6-7-2).
--------------------
10 => 3 = 168 [6]: (10-24-6-7-2-3).
--------------------
10 => 4 = 197 [7]: (10-24-6-7-2-3-4).
--------------------
10 => 5 = 162 [6]: (10-24-6-7-8-5).
--------------------
10 => 6 = 68 [3]: (10-24-6).
--------------------
10 => 7 = 99 [4]: (10-24-6-7).
--------------------
10 => 8 = 131 [5]: (10-24-6-7-8).
--------------------
10 => 9 = 59 [3]: (10-24-9).
--------------------
10 => 10 = 0 [2]: (10-10).
--------------------
10 => 11 = 27 [2]: (10-11).
--------------------
10 => 12 = 159 [6]: (10-24-6-7-8-12).
--------------------
10 => 13 = 187 [7]: (10-24-6-7-8-12-13).
--------------------
10 => 14 = 213 [8]: (10-24-6-7-8-12-13-14).
--------------------
10 => 15 = 215 [8]: (10-24-6-7-8-12-13-15).
--------------------
10 => 16 = 243 [9]: (10-24-6-7-8-12-13-15-16).
--------------------
10 => 17 = 243 [9]: (10-24-6-7-8-12-13-15-17).
--------------------
10 => 18 = 271 [10]: (10-24-6-7-8-12-13-15-17-18).
--------------------
10 => 19 = 242 [9]: (10-24-6-7-8-12-13-14-19).
--------------------
10 => 20 = 72 [3]: (10-21-20).
--------------------
10 => 21 = 42 [2]: (10-21).
--------------------
10 => 22 = 73 [3]: (10-21-22).
```

Рис. 2.17 Результат роботи функції getResults

3. ТЕСТУВАННЯ МОДЕЛІ

3.1. Підхід до тестування системи

Для тестування роботи різних компонентів існує багато підходів. Перевірка даної моделі системи проводиться за допомогою модульного тестування. Модульне (unit) тестування – це варіант перевірки коректності роботи ПЗ, що дозволяє окремо тестувати кожний модуль коду програми окремо один від одного, або з підключенням мінімальної кількості залежних модулів. Модуль – це мінімальна частина програми, що може бути перевірена.

Модульне тестування в мові PHP реалізоване бібліотекою PHPUnit. Ця бібліотека впроваджує такі можливості:

- попередня ініціалізація змінних, суміжних для усіх ситуацій перевірки;
- можливість друку інформації про перебіг перевірки в текстовому чи графічному режимі;
- можливість розкладання структури перевірочного покриття;
- можливість запуску деякої кількості тестів одночасно;
- та інші.

Через те, що основними об’єктами тестування у системі є розраховані маршрути руху для кожного транспортного засобу, треба представити істині дані, за якими PHPUnit буде перевіряти отримані результати. Таким чином подаємо істині дані у xml-файлі, через те що xml-формат легко сприймається людиною, а також у мові програмування є функціонал для обробки xml-файлів.

Тестування відбувається наступним чином (рис. 3.1).

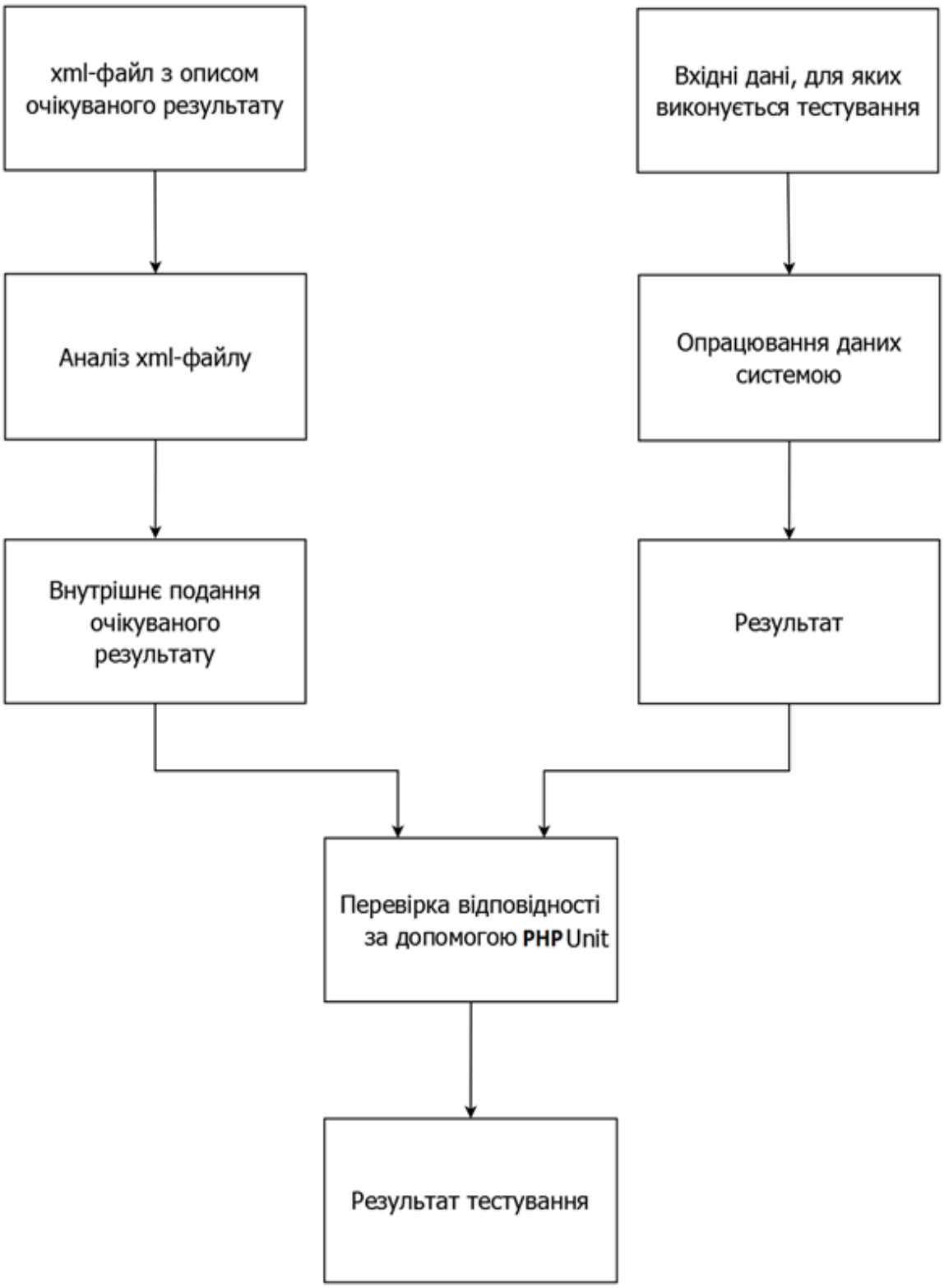


Рисунок 3.1 – Робота системи тестування

Спочатку в xml вводиться повна структура потрібної у коректних ситуаціях роботи програми, або очікуваного результату обробки даних моделлю. Потім отриманий xml-файл зчитується програмою-тестувальником і створюється внутрішнє представлення об’єктів моделей. Після цього запускається модуль, що ми перевіряєм для тих вхідних даних, які представлені результатом PHPUnit. В результаті, проводиться порівняння вхідного результату з отриманим.

3.2. Тестування моделі

Ефективність моделі системи аналізуватимемо за кривою збільшення часу прорахунку оптимального маршруту, якщо час обробки для відносно довгого маршруту та великої кількості транспортних засобів не перевищує декількох секунд, то робота системи є ефективною.

На рисунку зображено графік залежності часу обробки від кількості транспортних засобів.

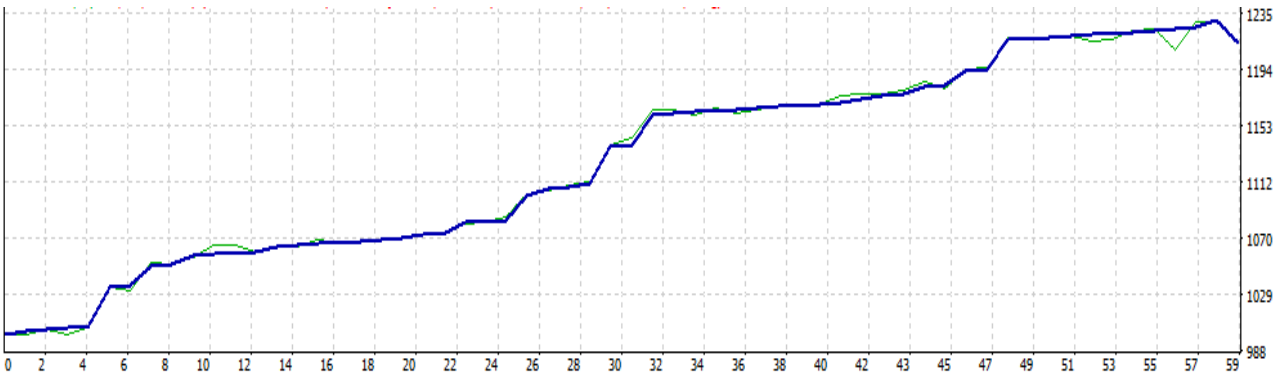


Рисунок 3.5 – графік залежності часу від кількості транспортних засобів.

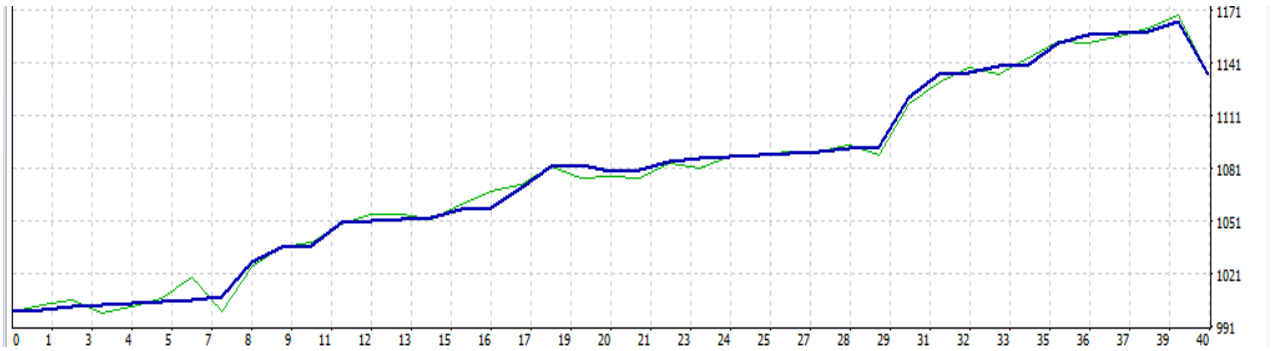


Рисунок 3.5 – графік залежності часу від відносної відстані.

Таким чином робота системи не сильно впливає на процесорний час процесора, та з моделлю можливо користуватись без відчутно довгого часу обробки модельованих ситуацій.

Час повного завантаження моделі зазвичай на кілька секунд більший ніж отриманий на тестах. Це цілком природньо. Причина в тому, що після розрахування оптимальних маршрутів результати передаються в контролер, після чого з отриманими даними генерується html та jQuery код, що буде відображати анімовану модель проїзду транспортних засобів, кожний за своїм маршрутом.

Також на час роботи моделі може вплинути кількість одночасних користувачів моделі, оскільки недостатньо потужному серверу доведеться паралельно прораховувати одразу кілька незалежних моделей. Робота графічного процесора малої потужності у користувача може вплинути на недостатньо плавну анімацію руху, та створити ефект, так званих «фризів», або короткочасних зупинок анімації.

4. РОБОТА ПРОГРАМИ

В цьому розділі розглядається інтерфейс програми та яким чином відбувається спілкування користувача з моделлю.

4.1. Алгоритм взаємодії

Моделювання роботи системи автоматизованої логістики транспортних засобів в програмі відбувається за наступним алгоритмом:

- Користувач моделі заходить за адресою через браузер з будь-якого пристрою, що має зв'язок з мережею Інтернет.
- Далі потрібно обрати режим моделювання чи інші функції в меню (рисунок 4.1).
- Після натискання на необхідний пункт меню відбувається завантаження сторінки.
- Якщо обрано один з режимів моделі, то завантажується модель, якщо інші опції, то завантажується відповідно створення користувацької мапи, режим тестування навантаження, режим текстового виводу алгоритму Дейкстри для відповідної мапи.

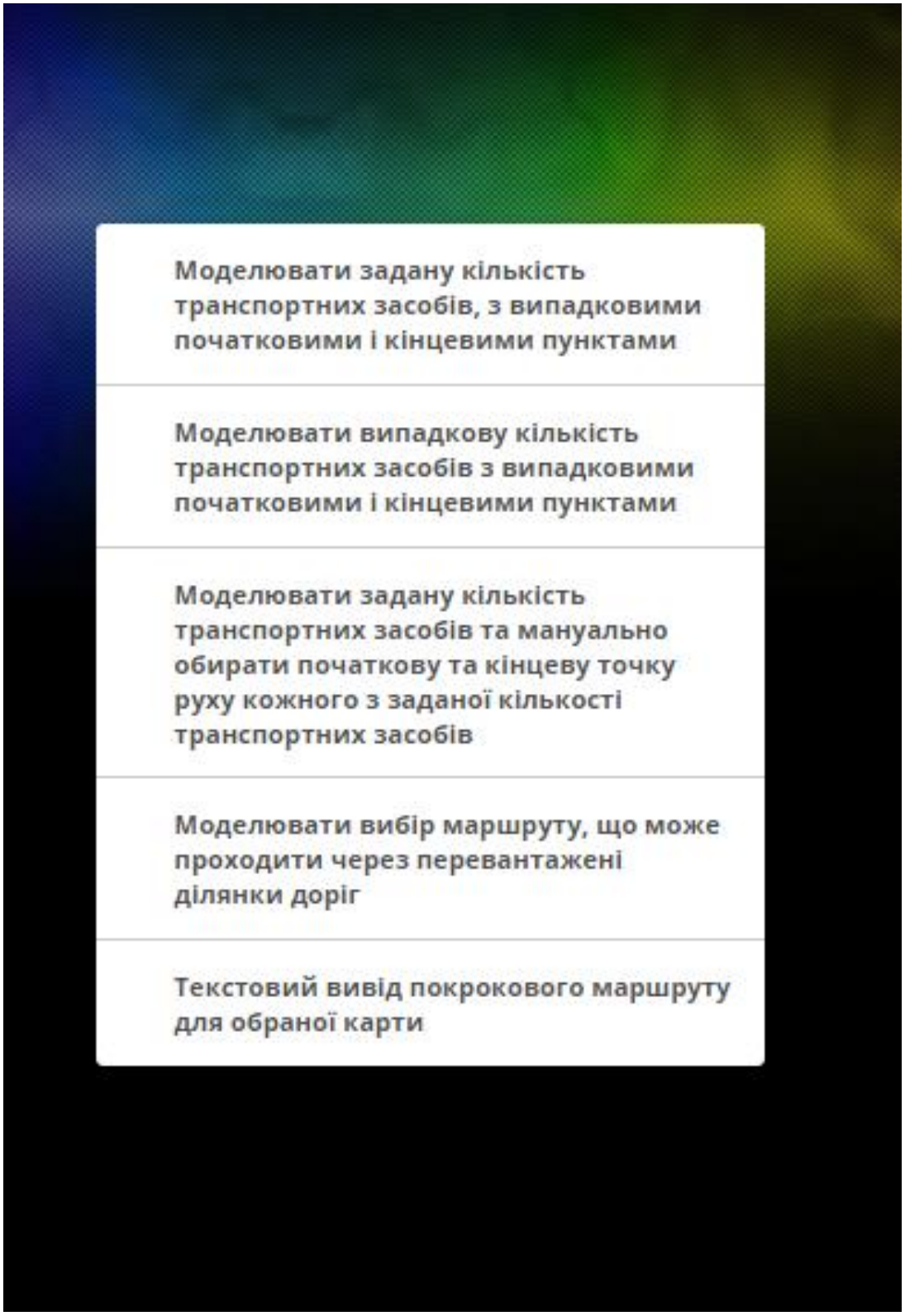


Рисунок 4.1 – Робота системи тестування

Для відтворення анімації необхідно ввести у формі усі необхідні параметри. Залежно від обраного режиму моделювання це можуть буди наступні поля:

- кількість автомобілів
- початковий та кінцевий пункт з представленої мапи для кожного транспортного засобу
- випадаючий список з варіантами вибору мапи, з вбудованих в програму, або зі створених користувачем
- поле для вибору чи виводити покроковий маршрут для кожного транспортного засобу.

4.2. Зовнішній вигляд програми

Зовнішній вигляд меню представлений на рисунку 4.1. У цьому меню користувач може обрати наступні режими моделювання:

- Моделювати задану кількість транспортних засобів, з випадковими початковими і кінцевими пунктами
- Моделювати випадкову кількість транспортних засобів з випадковими початковими і кінцевими пунктами
- Моделювати задану кількість транспортних засобів та мануально обирати початкову та кінцеву точку руху кожного з заданої кількості транспортних засобів
- Моделювати вибір маршруту, що може проходити через перевантажені ділянки доріг
- Текстовий вивід покрокового маршруту для обраної карти

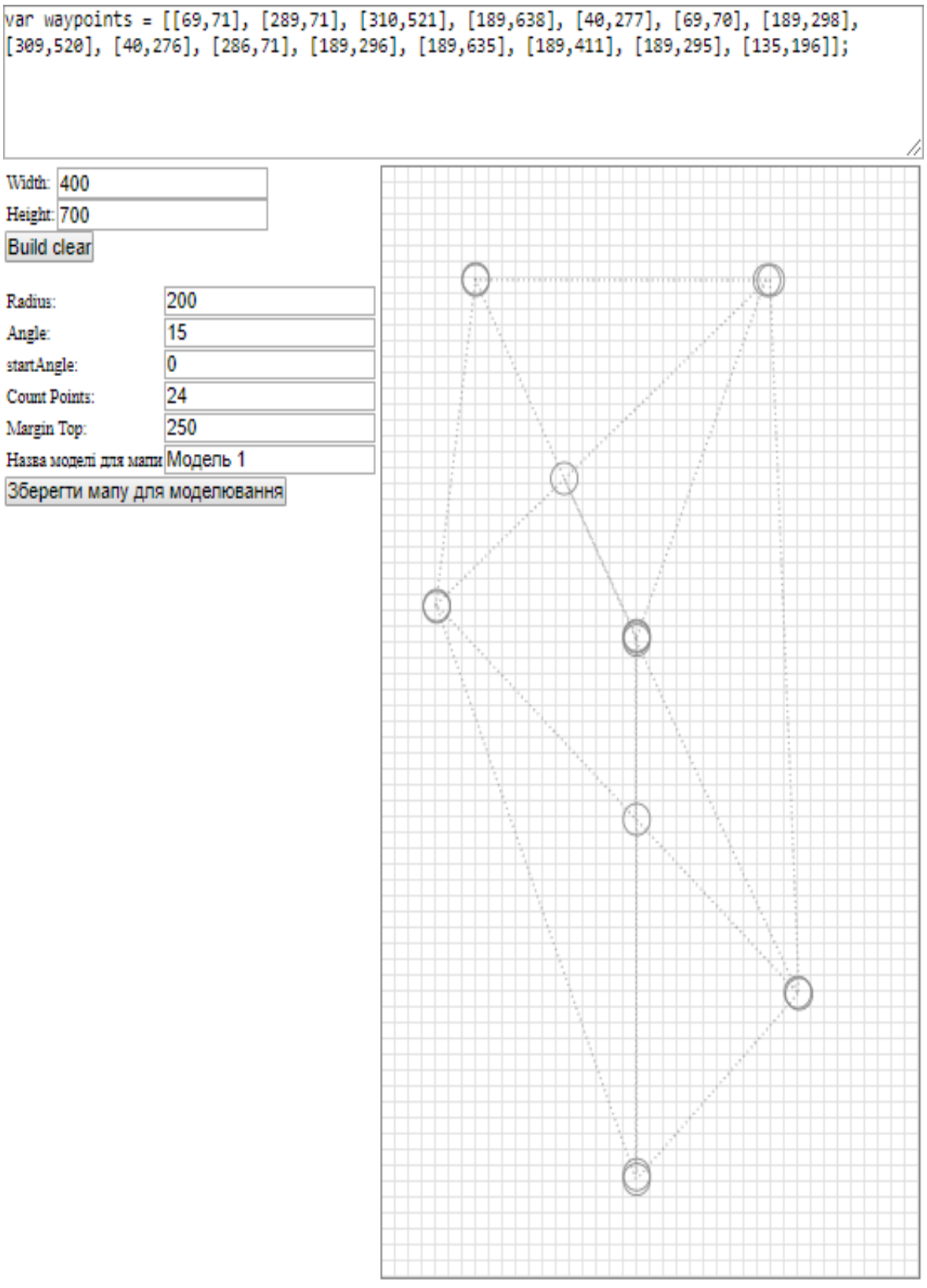


Рисунок 4.2 – Будувач мап для створення користувацької моделі

На рисунку 4.2 можна побачити як виглядає модуль створювача та редагувача мап для користувацької моделі.

Згори поле з координатами та зв’язками мапи, яке змінюється автоматично, коли користувач натискає на потрібні місця на зображенні з правого боку знизу.

Зліва форма для задання необхідних параметрів моделі та мапи. Знизу форми є поле для вводу імені моделі та кнопка для збереження моделі.

Зображення мапи обирається та завантажується користувачем.

На рисунку 4.3 зображено модель можливої користувацької мапи з відміченими точками на перехрестях. У цій моделі обрано 5 транспортних засобів, що переміщуються по мапі за заданим маршрутом. Маршрут розраховано з використанням алгоритму Дейкстри з реалізованою логікою покращення оптимального маршруту, описаного у попередніх розділах.

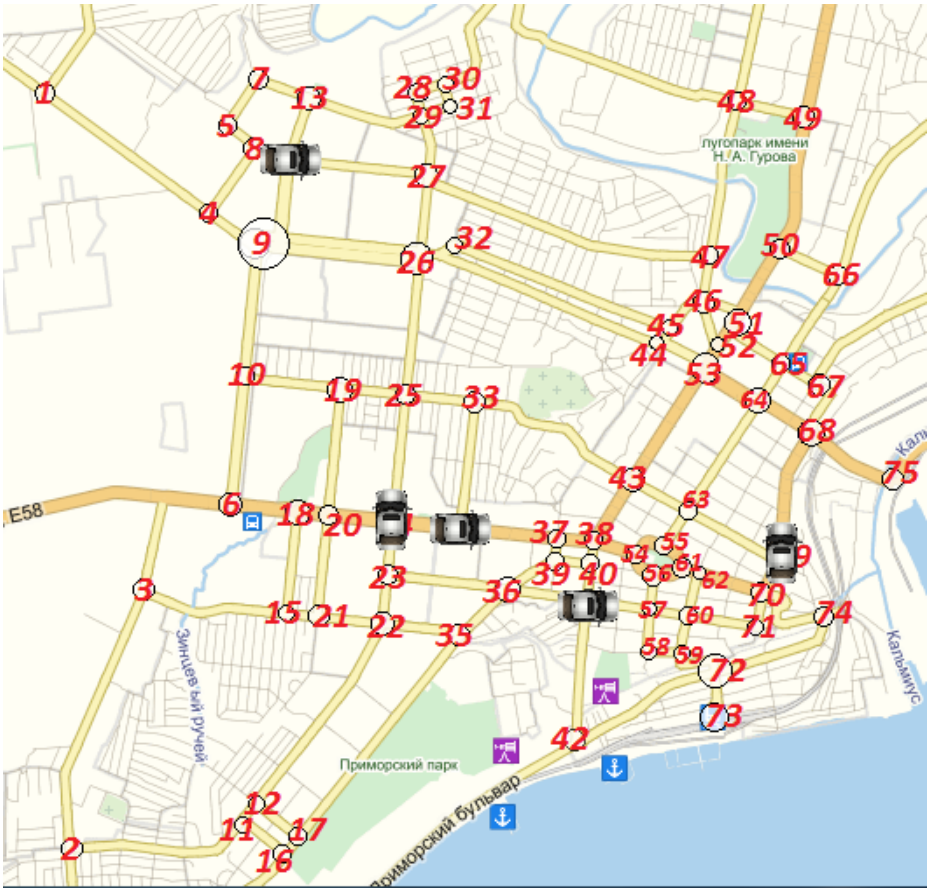


Рисунок 4.3 – Модель мапи з транспортними засобами що переміщуються

ВИСНОВКИ

Отже, за час роботи над дипломним проектом була створена автоматизована інтелектуальна система логістики транспортних засобів. При створені даної системи були використані сучасні алгоритми технічного аналізу, обробки даних, та алгоритмів прийняття рішення спроектованих на платформі Laravel завдяки мові програмування PHP.

Розроблена модель системи є ефективнішою за аналоги, та в порівняні з людиною, оскільки проводиться аналіз не тільки окремих параметрів, що не змінюються досить тривалий час, а ще й динамічно визначає зміни в параметрах, приймаючи до уваги як інші автомобілі, та їхні маршрути, так і зовнішні параметри, які можуть впливати на роботу системи з розрахунком маршрутів.

Цей проект може зацікавити не тільки студентів а й спеціалістів технічної області, дана система та її модель можуть бути основою для створення більш складних систем завдяки гнучкості, та логічному розбиттю на окремі модулі зі своїми функціями.

Ця система є втіленням усіх знань отриманих в процесі навчання в університеті, та добре відображає різноманітність та якість підготовки технічних спеціалістів. В цілому автоматизована інтелектуальна система виконана повністю з відповідністю до задачі дипломного проекту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Задача про найкоротший шлях — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Задача_про_найкоротший_шлях

2. Алгоритм Дейкстри — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_Дейкстри

3. Алгоритм Беллмана–Форда — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_Беллмана—Форда

4. Алгоритм пошуку A* — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_пошуку_A*

5. Алгоритм Флойда–Воршелла — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_Флойда_—_Воршелла

6. Алгоритм Джонсона — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_Джонсона

7. PHP — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/PHP

8. Laravel — Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Laravel